23. The Finite Fourier Transform and the Fast Fourier Transform Algorithm

1. Introduction: Fourier Series

Early in the Nineteenth Century, Fourier, in studying sound and oscillatory motion conceived of the idea of representing periodic functions by their coefficients in an expansion as a sum of sines and cosines rather than their values.

He noticed that if, for example, you represented the shape of a vibrating string of length L, fixed at its ends as

$$y(x) = \sum a_k \sin 2 \pi kx/L,$$

the coefficients, a_k, contained important and useful information about the quality of the sound that the string produces that was not easily accessible from the ordinary y=f(x) representation of the shape of the string.

This kind of representation of a function is called a Fourier Series, and there is a tremendous amount of mathematical lore about properties of such series and for what classes of functions they can be shown to exist.

One particularly useful fact about them is how we can obtain the coefficients a_k from the function.

This follows from the orthogonality property of sines:

$$\int \sin 2 \pi kx/L \sin 2 \pi jx/L \, dx$$

if the integral has limits 0 and L, is 0 if k is different from j and is L/2 when k is j.

(To see this notice that the product of these sines can be written as a constant multiple of the difference between cosines of $2\pi(k+j)x/L$ and $2\pi(k-j)x/L$, and each of these cosines has 0 integral over this range.)

By multiplying the expression for y(x) above by $2\pi jx/L$ and integrating the result from 0 to L we get then the expression

$$a_j = (1/\pi) \int f(x) \sin 2\pi jx/L \, dx.$$

Fourier series represent only one of many alternate ways we can represent a function. Whenever we can, by introducing an appropriate weight function in the integral, obtain a similar orthogonality relation among functions, we can derive similar formulae for coefficients in a series.

2. The Fourier Transform

Given a function f defined for all real arguments, we can give an alternative representation to it as an integral rather than as an infinite series, as follows.

$$f(x) = \int \exp(ikx) \, g(k) \, dk$$

where the integral is over all real values of k.

The representation of f by the function g is called a Fourier transform of f, and it is very important tool in physics.

One reason for this is that exponential functions e^{ikx}, which f is written as an integral which is a sort of a sum of are eigenfunctions of the derivative. That is, the derivative, acting on an exponential, merely multiplies the exponential by ik. This makes the Fourier transform a useful tool in investigating differential equations.

Another example of it's application: In quantum mechanics, we represent the state of a particle in a physical system has wave function, ?(x), and $|?(x)|^2$
dx represents the probability that the particle in this state has position that lies between x and x+dx.

The same state can also be represented by its wave function in momentum space, and that wave function of the variable p, is a constant multiple of the Fourier transform of φ(x):

$$?(p) = c \int \exp(ipx)\varphi(x)dx.$$

We can invert the Fourier Transform in much the same way that we can invert Fourier Series. The resulting formula is

$$g(k) = (1/2\pi) \int \exp(-ikx)\, f(x)\, dx$$

again the integration is over all real values of x.

3. The Finite Fourier Transform

Given a finite sequence consisting of n numbers, for example the ccoefficients of a polynomial of degree n-1, we can define a Finite Fourier Transform that produces a different set of n numbers, in a way that has a close relationship to the Fourier Transform just mentioned.

I like to look at it backwards.

Suppose we have a polynomial p of degree n-1. It can be described by its coefficients, $\{a_j\}$: with

$$p(x) = ?_{j= 0 \text{ to } n-1}\ a_j\ x^j\ .$$

We can also represent p by giving its values at any n arguments $\{p(x_k)\}$.

This can be done as follows. Observe first that the polynomial of degree n-1

$$f(x_j)((x-x_1)\ /(x_j-x_1))*\ .\ .\ .(\text{exclude } (x-x_j)\ /(x_j-x_j))\ .\ .\ .\ *((x-x_n)/\ (x_j-x_n))$$

takes the value $f(x_j)$ at $x=x_j$ and is 0 at all other of our arguments x_k.

We can recover p from its values by summing similar terms over all j.

To evaluate a polynomial of degree n-1 at n values appears to require n^2 actions: n evaluations each of n terms. Similarly the procedure just described for recovering p from its values requires at least n^2 operations to obtain all the coefficients of p.

When you evaluate a polynomial at an argument x whose magnitude is not close to 1, the powers of x that are big dominate those that are small by so much that you have to worry about losing the smaller terms entirely from round off errors.

The finite Fourier transform can be defined as the act of evaluating a polynomial of degree n-1 at n roots of unity, that is, at n solutions to the equation $x^n=1$.

This transform can be performed upon polynomials with coefficients in any field in which this equation has n solutions, which will happen when there is **a primitive n-th root of unity** in the field. (This means a number such that $x^n=1$ but x^k
is not 1 for any k between 1 and n-1.) The n roots of unity are then the various powers of the primitive root.

When does this happen? It does for complex numbers, in which case we have

$$\mathbf{\exp(2\pi i/n)} \text{ which is } \mathbf{\cos(2\pi/n)+i\sin(2\pi/n).}$$

as primitive n-th root of unity.

But it also happens for remainders on dividing by a prime number of the form kn+1. In such fields there is a primitive kn-th root of unity and hence a primitive n-th root of unity (such as the k-th power of the former.)

The analogy between this finite transform and the Fourier transform is mnost apparent when we use complex numbers. Then, if the coefficients of the polynomial are $\{a_j\}$, the evaluations become

$$p(\exp(2\pi ik/n)) = ?_j\, a_j\, (\cos(2\pi jk/n)) + i?_j\, a_j\, (\sin(2\pi jk/n)).$$

(This is why we say that we are doing things backwards. It is the a_j which are analogous to the Fourier coefficients for the function p.)

In general, if we let z be our primitive n-th root of unity, the same expression becomes

Transforms of this kind can be defined for any value of n. And there is a symmetric form for the inverse transformation which takes the values

$\{p(z^k)\}$, which we shall abbreviate as $\{p_k\}$, and produces the a_j, so there is no significant difference between defining this transform forward or backward.

We can obtain the inverse transformation by multiplying each p_k by z^{-sk},

and summing over the n values of k.

We get $?_{j,k}\, a_j\, z^{jk} * z^{-sk}$ or $?_j\, a_j\, (\Sigma_k\, z^{(j-s)k})$ or $\Sigma_j\, a_j\, t_{s-j}$, where t_r is our old friend the sum of the r-th powers of the n roots, z^k, of the equation $z^n - 1$.

Recall please, that this equation, $z^n - 1 = 0$ has the form $\Sigma_k\, z^k s_k = 0$, where s_k is the k-th elementary symmetric function of the roots of the equation.

This implies that the s_k are all 0, for k=1 up to n-1 for our equation, while s_0 is 1.

Recall also that the the t's and the s's are linearly dependent according to the relations, for each k

$$\Sigma_{j=0 \text{ to } k-1}\, s_j t_{k-j} (-1)^j + k s_k (-1)^k = 0,$$

from which we can deduce that the t_{s-j} here are all 0, unless s=j.

When s=j, the sum that forms t_0, or t_n, which is n, so that we get

$$\Sigma_k\, p_k\, z^{-k} = \Sigma_{j,k}\, a_j\, z^{jk} * z^{-sk} = n a_s,$$

and

$$a_s = (1/n) \Sigma_k p_k z^{-k}.$$

Since z^{-1}
is another primitive n-th root of 1 the only difference in this equation, and the inverse equation for evaluations is in the factor 1/n.

4. The Cooley-Tukey Fast Fourier Transform Algorithm

Suppose n is even, so that n can be written as 2s. Then this algorithm is a procedure for reducing 2s evaluations of polynomials of degree 2s to 2s evaluations of polynomials of degree s, upon making a total of s additions, s subtractions and s multiplications. Moreover the evaluations consist of evaluating the FFT's of two polynomials, each of degree s-1, at primitive s-th roots of unity.

To keep things straight let us describe the evaluations of a polynomial of degree at most n-1 at n n-th roots of unity.as an nFFT.

If n is a power of 2, we can iterate this procedure n times, until we reduce the problem to n evaluations of polynomials of degree 0, which is a weird way to say that we will have obtained our n evaluations.

The reduction that is the heart of this algorithm is based upon the following observations. To perform an nFFT require evaluating our polynomial of degree up to n-1 at the n powers of a primitive n-th root of unity, z.

1. If we consider the evaluations we seek at the even powers of z, $(1, z^2, z^4, \ldots)$, these powers are the powers 0 to s-1 of the s-th root of unity z^2.

Thus, these evaluations are exactly what is involved in an sFFT; the only difference being that we are here evaluating a polynomial of degree up to n-1, not up to s-1.

2. In every even power evaluation, say at z^{2k}, the term in our polynomial $a_j x^j$ contributes $a_j z^{2kj}$. Thus, the contribution from the j-th and (s+j)-th terms together are

$$a_j z^{2kj} + a_{j+s} z^{2kj+2sk}.$$

But, z^{2sk} is z^{nk} which is 1, so that the a_{j+s} contribution here is multiplied by the same power of z as the a_j contribution, and can be added to it instead of being treated as a separate term.

3. **But this means that the z^{2k} evaluations here are exactly those of sFFT($\{a_j+a_{j+s}\}$).**

4. The odd power evaluations, (at z, z^3, . . .) each gets a contribution from a_j of the form $a_jz^{(2k+1)j}$ which we can write as $(a_jz^j)z^{2jk}$. Notice
that these evaluations can be considered evaluations at even roots of unity of a polynomial whose coefficients are given by the products (a_jz^j).

5. In every odd power evalution, say at z^{2k+1}, the term in our polynomial a_jx^j contributes $a_jz^jz^{2kj}$ while the term $a_{j+s}x^{j+s}$ contributes $a_{j+s}z^jz^sz^{2kj\ +2ks}$. As in our second observation we have $z^{2ks}=1$, but now we have an additional factor of z^s, which is a primitive square root of 1, which is –1. In other words the contributions from a_j and $(-a_{j+s})$ are the same here.

6. We conclude that
we can combine the j and s+j terms by subtraction in the odd power evaluations and these become exactly those of sFFT($\{(a_j-a_{j+s})z^j\}$).

You will notice that we make an addition for each of our even power evaluations in making these reductions, and a subtraction and a multiplication for each of the odd powers reductions, and this means that we must perform s of each of these operations to reduce the problem by a factor of two.

The remaining task in completing our evaluations after this reduction consists of repeating it in parallel for the even and odd evaluations. After a second reduction, we perform 4 reductions in parallel for the even-even, odd-even even-odd and odd-odd evaluations (starting in positions z^0,z^1, z^2,z^4), and so on.

And that is the algorithm.

We illustrate it by starting with 4 coefficients 1,3,2,5 (of powers 0 through 3) and do our calculations mod 17. 4 is a primitive 4^{th} root of 1 whose inverse is –4 or 13.

In the first step we replace the even power entries, which are the first and third here, by 1+2 and 3+5 respectively,

and the second and fourth entries by $(1-2)*4^0$ and $(3-5)*4^1$ which produces the sequence 3,16,8,9 mod 17.

In the second step we replace the first **and second**
entries by 3+8 or 11 and 16+9 or 8, and the third and fourth by 3-8 or 12 and 16-9 or 7, for an answer of (11,8,12,7) as the result of evaluating the polynomial $1+3x+2x^2+5x^3$ at x= 1,4,16 and 13 mod 17.

It is instructive to see what happens when we apply the same procedure again to the sequence obtained here, namely (11,8,12,7).

In the first step this becomes (6,16,15,4) (the last value comes from (8-7)*4 mod 17). And in the last step we get (4,3,8,12).

Notice that if we divide this result by 4 we get (1,5,2,3) (to divide 3 by 4 you can add 17 to the 3, divide 20 by 4 and get 5).

These are the original coefficients of our polynomial, in the order 0,3,2,1.

The reason for this is that the formula for the inverse of our transformation requires dividing by n and also using z^{-1} in place of z as the primitive n-th root of unity at whose powers the evaluations are made.

And of course evaluating at z^{-k} is the same thing as evaluating at $z^{(n-k)}$, which means the k-th power evaluation for z^{-1} is the n-k-th for z.